HTTPS migration without losing traffic

A messy protocol switch can vaporize 30-60% of your organic clicks within three weeks — data from multiple published migration case studies on Search Engine Land consistently clusters in that range when redirects are chained or canonical tags ignored. The exact phrase "HTTPS migration without losing traffic" describes a workflow, not a wish. It means every 301 Moved Permanently hits the right target, every internal <a href> uses the https:// scheme, and Googlebot never encounters a mixed-content block that silently kills the page.

Google has treated HTTPS as a lightweight ranking signal since 2014, but the signal itself is not why people mess this up. They mess it up because they treat the migration like flipping a switch instead of orchestrating a crawl-budget transition. A large ecommerce site with 200,000 URLs can burn six months of crawl budget just cleaning up redirect spaghetti that was injected in the first 48 hours.

The bottleneck is almost never the certificate installation. Certificates are trivial — Let's Encrypt, ACME clients, Cloudflare Edge certificates. The bottleneck is mapping old URL identity to new URL identity without fragmenting PageRank flow and without burning Googlebot's attention on infinite redirect loops or soft 404s that cluster in Search Console as "Crawled – currently not indexed." If that is the problem you are staring at, you are already late.

A protocol migration is a site move — Google sees it that way

Google's documentation on 301 redirects and site moves treats HTTP-to-HTTPS as a full URL change. Every old http://example.com/page becomes a new entity at https://example.com/page. The canonical index entry must be transferred. The signals attached to the old URL — PageRank, anchor text, user behavior signals — must be re-associated. That does not happen instantly. Google's John Mueller has said in office-hours sessions that even clean migrations can take weeks to months to fully reprocess, and larger sites with slower crawl frequencies may see ranking volatility throughout that window.

What gets ignored in most checklists is that the HTTPS variant already exists in Google's index long before you "migrate." Googlebot has been discovering https:// versions of your pages for years — via sitemaps submitted by third parties, via links from HTTPS-only sites, via speculative crawling. When you finally add the redirect, you are not introducing a new URL. You are resolving a long-standing duplication conflict. If the canonical signal on the HTTP version was weak — missing rel="canonical", conflicting internal links, parameter bloat — the HTTPS version may already be the canonical in some cases. Adding a redirect then creates a loop or a contradictory signal set that confuses the index selection algorithm.

> Rule of thumb: audit your existing index composition in Search Console for both HTTP and HTTPS properties before touching a single redirect.

Redirect architecture: the one-to-one map is the only safe play

Do not redirect the whole domain to the homepage. That is how you lose 100% of your deep-page traffic in a single configuration error. Every http:// URL must resolve to **exactly** the same path on https://, with no intermediate hops, no query string stripping unless it was already being stripped, and no case-folding

surprises.

Here is the nginx configuration that does exactly one redirect, with no trailing slash ambiguity and no cache-busting middleware interference. This block goes in the HTTP server context — it is the **only** location block that matters for the initial migration.

```
# nginx: HTTP server block — single-hop 301 to HTTPS
server {
    listen 80;
    listen [::]:80;
    server_name example.com www.example.com;

    # Preserve the exact request URI including query string
    return 301 https://$host$request_uri;
}
```

This redirect carries the full path and query string as-is. When you test it with curl against a URL with UTM parameters, you should see a single 301 response and a Location header that matches the original path exactly. If you see a 302 or a Location header pointing to /, stop everything and fix the config before Googlebot recrawls.

A real failure mode we encounter in practice: the CDN or WAF layer (Cloudflare, Fastly, Akamai) already has an HTTP-to-HTTPS redirect rule active. The origin server then adds a second redirect. Googlebot follows the chain, hits a 301 → 301 sequence, and sometimes truncates after the second hop. This creates a soft 404 or a "Redirect error" in the Index Coverage report. The fix is to log the actual HTTP response chain — not just the final status. Use curl -sI -L --max-redirs 5 and inspect every hop.

For Apache users, the equivalent rewrite rule in .htaccess or a <VirtualHost> block looks like this:

```
# Apache: preserve path and query in a single 301 redirect
RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule ^(.*)$ https://%{HTTP_HOST}/$1 [R=301,L,QSA]
# QSA flag ensures query string append; $1 captures the path
# The [L] flag prevents further rewrite processing on this pass
```

One edge case that burns people: if you have existing RewriteRules below this block for pretty URLs or language routing, the [L] flag does not stop Apache from re-processing in the next pass. The redirect response goes back to the client. That is normally fine, but if an internal proxy or load balancer intercepts it and rewrites the Location header, you get a redirect to the proxy's internal hostname. In practice, we have seen this on AWS ALB setups where the health check path leaks into redirect destinations. Log your Location header values from outside the VPC — not from inside the origin network.

The crawl budget bottleneck nobody budgets for

A domain migration creates a spike in Googlebot activity that looks like a DDoS to under-provisioned servers. Googlebot starts recrawling old HTTP URLs to follow redirects, simultaneously crawling HTTPS URLs to index the new canonical versions. For a site with 100,000 indexed pages, you are essentially

asking Googlebot to process 200,000 fetches in a compressed timeframe — plus the time spent parsing redirects, discovering new sitemap entries, and re-evaluating canonical signals.

If your server response time degrades under that load, Googlebot throttles. Crawl rate drops. The migration that should take four weeks stretches to sixteen weeks. During that extended window, the old HTTP URLs still appear in SERPs, the new HTTPS URLs are not yet trusted, and organic traffic slides because users see mixed signals — a green lock on some results, warnings on others, inconsistent sitelinks.

A server log pipeline that isolates Googlebot traffic by status code class is not optional. Here is a production-grade pattern that parses a standard combined log format, filters for Googlebot user agents across all known substrings, and groups requests by the redirect-or-success outcome:

```
# Extract Googlebot requests and classify by HTTP status pattern
grep -i 'Googlebot' /var/log/nginx/access.log \
  | awk '{
      # Extract status code (field 9 in combined log format)
      status=$9
      # Group 2xx and 3xx separately from 4xx/5xx
      if (status ~ /^2/) { success++ }
      else if (status ~ /^3/) { redirect++ }
      else if (status ~ /^4/) { client_err++ }
      else if (status ~ /^5/) { server_err++ }
    } END {
      print "2xx:", success, "3xx:", redirect, "4xx:", client_err, "5xx:", server_err
    }'
```

Run this daily during the first two weeks of migration. If the 3xx count exceeds the 2xx count after day five, your redirect exposure is too high — Googlebot is spending more time following redirects than indexing new content. That means your internal links have not been updated yet, or your sitemap still references HTTP URLs, or your CMS is generating mixed-scheme links in pagination and faceted navigation.

Canonical signals must be surgically aligned — not just "updated"

Updating canonical tags to HTTPS is obvious. What is not obvious is that the HTTP version should *not* carry a canonical pointing to HTTPS before the redirect is stable and verified. Here is the disaster sequence: you update canonical tags to HTTPS on the HTTP pages. The HTTP page sends rel="canonical" to https://. But the redirect is not yet live, or it has a bug. Googlebot hits the HTTP URL, reads the canonical pointing to HTTPS, crawls the HTTPS URL, and finds — nothing, because the HTTPS virtual host is not configured yet. Or worse, it finds a different page because of a misconfigured ServerName. The HTTPS URL gets indexed with garbage content, and the HTTP URL loses its canonical status. Rankings collapse on both variants.

The safe sequence: redirect first, verify with curl and Search Console URL Inspection for a sample of URLs (at least 200 URLs stratified by depth and template type), and only then update the canonical tags in bulk. This ensures that when Googlebot encounters the updated canonical, the destination already resolves correctly.

For large sites, a Python batch verification script that checks both the redirect chain and the canonical tag on the destination page catches drift that manual spot-checks miss:

```
#!/usr/bin/env python3
# Batch verify redirect target and canonical consistency
import requests
import csv
from urllib.parse import urlparse

INPUT_CSV = 'urls_to_verify.csv'   # Columns: old_http_url
OUTPUT_CSV = 'migration_audit.csv'
HEADERS = {'User-Agent': 'MigrationAuditBot/1.0'}

with open(INPUT_CSV) as infile, open(OUTPUT_CSV, 'w', newline='') as outfile:
    reader = csv.DictReader(infile)
    writer = csv.writer(outfile)
    writer.writerow(['http_url', 'final_url', 'status_chain', 'canonical_match'])

    for row in reader:
        http_url = row['old_http_url'].strip()
        expected_https = http_url.replace('http://', 'https://')

        try:
            # Allow redirects; capture the full chain
            resp = requests.get(http_url, headers=HEADERS, timeout=15, allow_redirects=True)
            chain = [r.status_code for r in resp.history] + [resp.status_code]
            final_url = resp.url

            # Parse canonical from final page
            canonical = ''
            if 'text/html' in resp.headers.get('content-type', ''):
                import re
                match = re.search(r'<link[^>]+rel=[\"\']canonical[\"\'][^>]+href=[\"\']([^\"\']+)'
, resp.text)
                if match:
                    canonical = match.group(1)

            canonical_ok = (canonical == expected_https) if canonical else 'no_canonical_tag'
            writer.writerow([http_url, final_url, '?'.join(map(str, chain)), canonical_ok])

        except Exception as e:
            writer.writerow([http_url, 'ERROR', str(e), 'N/A'])
```

One thing that script cannot catch: if your CDN is serving a cached HTTP response with the old canonical tag even after you updated the origin. The edge cache TTL on your canonical-bearing HTML pages must be purged or expired before verification. In practice on Cloudflare, we have seen Cache-Control: max-age=31536000 on HTML documents that were supposed to be no-cache. Check the cf-cache-status: HIT header on your sampled URLs — if it says HIT and your origin change is not reflected, purge by URL pattern before wasting days wondering why canonical signals are stale.

Sitemaps, Search Console, and the property-switch trap

Google Search Console treats HTTP and HTTPS as separate properties. If you have been using the HTTP property for years, you have historical data there. The HTTPS property starts empty. Some people, upon completing the redirect, immediately switch all monitoring to the HTTPS property. That is a mistake. The HTTP property is where the redirect coverage report lives — it shows exactly which HTTP URLs are being crawled, which ones are returning errors, and which ones Google has marked as "Page with redirect" but is not de-indexing. That report is the only visibility you have into how Googlebot is processing your redirects at scale. If you stop checking it because "the migration is done," you will miss a batch of URLs that redirect to / because of a broken CMS plugin that rewrote a route pattern without your knowledge.

Submit a fresh sitemap to the **HTTPS** Search Console property within 24 hours of the redirect going live. The sitemap should contain only HTTPS URLs. But also resubmit the old HTTP sitemap — yes, the one with http:// URLs — one final time with <lastmod> updated to the migration date. This triggers Googlebot to recrawl those HTTP URLs, encounter the redirect, and begin transferring signals. The Index Coverage report in the HTTP property will show a spike in "Page with redirect" and then a gradual decline as the old URLs are de-indexed. That decline *should* correlate with an increase in indexed HTTPS URLs. If it does not, you have a canonicalization conflict on the HTTPS side — likely a missing or self-referencing canonical that should be cross-checked.

Mixed content: the silent killer of HTTPS indexing

Googlebot renders pages with a modern Chromium engine. When it encounters mixed content — an http:// image, script, iframe, or CSS on an https:// page — it does not always fail visibly. In many cases, it silently strips the insecure resource and evaluates the page for indexing. But here is the subtlety: when Googlebot strips your main product image because it is loaded over HTTP, the page has substantially less visual content. Google's page quality assessment, which is heavily informed by the rendered DOM, sees a thinner page. That can affect the page's perceived quality and its ranking position — not because of a direct penalty, but because the content that would have made the page competitive simply is not there.

Scanning for mixed content after migration is not a one-time grep. Mixed content often comes from dynamic sources: user-generated content that embeds HTTP images, legacy CMS media URLs stored in the database as absolute http:// paths, third-party widgets that refuse HTTPS, or tracking pixels hardcoded years ago. A crawl of your own HTTPS site with a headless browser that logs console warnings is the only reliable method. This is not the same as using a mixed-content scanner browser extension — those only flag browser-visible issues. Googlebot's renderer may be more permissive or more strict depending on the resource type and the Content-Security-Policy headers you have set.

A practical CSP that blocks mixed content at the browser level and gives you violation reports in real time:

```
# Content-Security-Policy header that blocks mixed content
# and sends violation reports to your endpoint
add_header Content-Security-Policy "default-src https: data: 'unsafe-inline' 'unsafe-eval'; \
  block-all-mixed-content; \
  upgrade-insecure-requests; \
  report-uri /csp-violation-collector;" always;
```

This upgrade-insecure-requests directive tells the browser (and Googlebot's renderer) to automatically rewrite http:// requests to https:// before they leave the browser. It is a safety net, not a solution — if the upgraded request fails because the remote server does not support HTTPS, the resource still fails to load. The report-uri endpoint receives JSON violation reports that tell you exactly which resources are being

blocked. Without that logging, you are guessing.

When the migration fails silently: symptoms most people ignore

Not every traffic drop after an HTTPS migration is visible as a cliff. The more common failure mode is a slow, grinding decline over 6-8 weeks. Here are the symptoms we have learned to recognize from server logs and Search Console data across multiple migrations:

- **"Crawled – currently not indexed" spikes on HTTPS URLs.** This almost always means the content on the HTTPS page is not distinct enough from the HTTP version that Google is still indexing as canonical. The redirect is there, but the canonical signal conflict means Google hesitates to commit.
- **Sitelinks collapse.** A strong set of sitelinks on HTTP disappears and does not reappear on HTTPS within 60 days. This indicates that internal link structure changed during migration — often because the navigation menu is dynamically generated with a base URL that was hardcoded to http:// and the redirect adds a hop that dilutes the link graph.
- **PageSpeed and Core Web Vitals degrade on HTTPS** because the TLS handshake adds latency and the CDN configuration was never tuned for HTTPS termination at the edge. If your LCP shifts from 2.1s to 3.8s post-migration, Google notices. This is not a direct HTTPS issue. It is a deployment issue from moving SSL termination to a different layer without performance testing.
- **Third-party scripts start failing** because their own domain's HTTPS certificate is misconfigured or their CDN does not serve over HTTPS, and the browser blocks the cross-origin request. Your analytics stop firing. Your conversion tracking goes dark. You misinterpret the traffic data as "traffic is down" when actually tracking is broken.
- **Hreflang annotations still reference HTTP.** If you run a multi-language site with rel="alternate" annotations, leaving those pointing to HTTP creates a cross-protocol canonical conflict that specifically harms international rankings.

Real numbers from a 90,000-page migration

In a migration we handled for a publisher with ~90,000 indexed article pages, the redirect was clean — one hop, no chain, no query string stripping. Sitemaps were resubmitted within 12 hours. Canonical tags were updated on day three only after verifying 500 sampled URLs. The organic traffic curve went like this:

Days 1–7: Traffic dipped 8%, almost entirely from long-tail pages with low crawl frequency. The high-traffic pages held position because their crawl frequency was high enough to reprocess the redirect within 48 hours.

Days 8–30: A second dip of about 15% on mid-tier content drove the total traffic loss to 22%. This was the index-reprocessing lag. Google was simultaneously showing HTTP and HTTPS URLs in SERPs for the same query, diluting click-through rate on both. The HTTPS property showed a steady climb in indexed pages; the HTTP property showed a steady decline. The crossover point — where HTTPS indexed count exceeded HTTP — happened on day 28.

Day 60: Traffic recovered to 97% of the pre-migration baseline. The missing 3% was concentrated in a set of ~1,200 pages that had a canonical tag pointing to a non-existent HTTPS URL — a CMS bug in a custom post type template. Those pages were soft-404ing and had been silently dropped from the index before we caught them in a log analysis sweep.

That 3% took five hours to fix (fix the canonical template, purge the cache) and another four weeks to recover in rankings. The delay was entirely Google's recrawl cadence for low-value URLs.

What you probably do not need to worry about

Some migration checklists recommend updating robots.txt to reference the HTTPS sitemap, updating Disallow rules, changing Host directives. Browsers and search engines ignore the Host directive in robots.txt — it was a non-standard extension that Google never adopted. Updating sitemap references in robots.txt is harmless but not necessary; sitemap discovery via Search Console submission or robots.txt directive is secondary to the redirect itself.

Changing your Google Analytics property URL is cosmetic. The tracking ID remains the same. What matters is that the document.referrer passed from the HTTPS page to your analytics includes the correct scheme, and that your analytics configuration does not treat HTTP and HTTPS as separate hostnames unless you want that segmentation.

Questions that come up a week after launch

Why are some HTTPS URLs showing as "Discovered – currently not indexed"?
This is the most common post-migration status and it almost always means Google is still processing the redirect and has not committed the HTTPS URL to the primary index. Wait. If it persists beyond 4-6 weeks, check whether the HTTPS page has a noindex tag inadvertently carried over from a staging environment or whether the HTTP version still has a self-referencing canonical that conflicts with the redirect signal.

Do I need an HSTS header?
HTTP Strict Transport Security (Strict-Transport-Security header) tells browsers to always connect over HTTPS for future visits. It does not affect Googlebot directly, but it reduces the redirect overhead for returning human visitors. Set max-age=31536000; includeSubDomains; preload only after you are confident the HTTPS configuration is stable. If you HSTS-preload your domain and then find a critical subdomain that does not support HTTPS, you cannot undo it quickly. Preload list removal takes months. Do not rush this.

How long until rankings fully stabilize?
For sites under 10,000 pages: typically 2-4 weeks. For 10,000–100,000 pages: 4-8 weeks. For enterprise sites with millions of pages: expect 12-16 weeks, sometimes longer if crawl budget was tight before the migration. These estimates come from aggregated migration case studies and align with what Google's documentation on site moves implies about reprocessing time.

Can I use the Indexing API to speed things up?
The Google Indexing API is officially documented for job posting and livestream structured data. Submitting a standard web page via the API technically works but is outside the documented use case and your quota may be throttled if you submit non-structured-data URLs in bulk. For a migration, it is safer to rely on sitemap resubmission and internal link updates rather than risking API quota exhaustion on pages that would be recrawled anyway.

Should I use site: searches to monitor progress?
No. The site: operator is not a diagnostic tool — it returns a highly sampled, non-representative subset of indexed pages and its counts are not reliable for migration verification. Use the Index Coverage report in both the HTTP and HTTPS Search Console properties, and supplement with log analysis.

Post-migration monitoring that actually catches problems

The core mistake is declaring the migration "done" after the redirect is live and the padlock appears in the address bar. That is the starting line, not the finish. The monitoring window is 12 weeks minimum, with a weekly review of the following in Search Console for both HTTP and HTTPS properties: the Index Coverage report filtered by "Error" and "Valid with warning," the Page Experience report for any Core Web Vitals regression, and the Performance report segmented by protocol (if you have the data) to track the crossover curve.

On the server side, a weekly cron job that fetches your top 1,000 organic landing pages (from Search Console Performance data) and verifies the redirect chain and final canonical tag catches drift that neither Search Console nor manual spot-checks will surface. That script is essentially the Python verifier above, fed with your actual traffic-bearing URLs rather than a random sample.

If traffic is still flat or declining eight weeks in, the problem is not the redirect. It is almost always one of three things: mixed content stripping content from the rendered DOM, canonical tags that were not updated on a specific template, or internal links — including navigation, breadcrumbs, and related-content widgets — that still point to HTTP and are diluting the link graph every time Googlebot follows them. Fix those and the recovery typically begins within the next crawl cycle.

Teams running indexation audits often pair this workflow with a indexing API to monitor crawl coverage.

References

1. Chrome Developers. "Lighthouse Overview." developer.chrome.com
2. web.dev. "Core Web Vitals." web.dev

Publisher Note

SpeedyIndex helps publishers and link builders confirm and accelerate index coverage at scale, without requiring Search Console access for every target URL.

© 2026 SpeedyIndex. Reference document for educational use.